LibRTOSConsole-C-Library

1.0

Generated on Sun Feb 16 2025 00:34:19 for LibRTOSConsole-C-Library by Doxygen 1.13.2

Sun Feb 16 2025 00:34:19

Chapter 1

FreeRTOS Console Library

1.1 Introduction

The following code documentation is a set of tipps and diagramms as well as function and structure descriptions which help when using the library. It is used to increase the implementation speed. Some examples are attached in this documentation as well

1.2 static compile flags of the library

CONSOLE_USERNAME: The username in the console which is printed at least on power up before changing the username
CONSOLE_USE_DYNAMIC_USERNAME: The username can be changed on runtime which is enabled with this DEFINE
CONSOLE_LINE_HISTORY: specifies the maximum number of entries of the "last commands" buffer
CONSOLE_LINE_SIZE: Specifies the number of chars per line. When exceeding this number without a newline, the console prints a buffer overflow and clears the input. All inputs are then invalidated within
this line
CONSOLE_COMMAND_MAX_LENGTH: Specifies the maximum number of chars per command
CONSOLE_HELP_MAX_LENGTH: Specifies the maximum number of chars per command help text

1.3 Examples

The following example shows how to create a instance of the console library and how to attach a function for console usage. This is called registering a command

```
static int CapabilityFunc( int argc, char** argv, void* ctx )
{
    printf("%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d\r\nOK",
        1, // has spindle
        1, // has spindle status
        1, // has stepper
        1, // has stepper move relative
        1, // has stepper move speed
        1, // has stepper move async
        1, // has stepper status
        1, // has stepper refrun
        1, // has stepper refrun timeout
        1, // has stepper refrun skip
        1, // has stepper refrun stay enabled
        1, // has stepper reset
        1, // has stepper position
        1, // has stepper config
        1, // has stepper config torque
        1, // has stepper config throvercurr
        1, // has stepper config powerena
        1, // has stepper config stepmode
        1, // has stepper config timeoff
        1, // has stepper config timeon
        1, // has stepper config timefast
        1, // has stepper config mmperturn
        1, // has stepper config posmax
        1, // has stepper config posmin
        1, // has stepper config posref
        1, // has stepper config stepsperturn
        1  // has stepper cancel
    );
    return 0;
}

...

// create the console processor. There are no additional arguments required because it uses stdin, stderr
      and
// stdout of the stdlib of the platform
ConsoleHandle_t c = CONSOLE_CreateInstance( 4*configMINIMAL_STACK_SIZE, configMAX_PRIORITIES - 5  );

// register the function, there is always a help text required, an empty string or null is not allowed!
CONSOLE_RegisterCommand(c, "capability", "prints a specified string of capability bits",
  CapabilityFunc, NULL);

...
```

The following example shows a really simple console function and how it is used to work with the console and arguments passed by the user

```
static int ConsoleFunction( int argc, char** argv, void* ctx )
{
  //possible commands are
  //(command) start
  //(command) stop

  // first decode the subcommand and all arguments
  if ( argc == 0 )
  {
    printf("invalid number of arguments\r\nFAIL");
    return -1;
  }
  if ( strcmp(argv[0], "stop") == 0 )
  {
    // do something
  }
  else if ( strcmp(argv[0], "start") == 0 )
  {
    // do something else
  }
  else
  {
    printf("invalid subcommand was given as argument\r\nFAIL");
  }
}
```

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Data Structure Documentation

4.1 cmdEntry Struct Reference

Data Fields

- struct {
  CONSOLE_CommandFunc **func**
  void ∗ **ctx**
  char **cmd** [CONSOLE_COMMAND_MAX_LENGTH+2]
  int **cmdLen**
  char **help** [CONSOLE_HELP_MAX_LENGTH+2]
  int **helpLen**
  } **content**

The documentation for this struct was generated from the following file:

- C:/HomeGit/STM32/libs/LibRTOSConsole/src/Console.c

4.2 cmdState Struct Reference

Data Fields

- SemaphoreHandle_t **lockGuard**

The documentation for this struct was generated from the following file:

- C:/HomeGit/STM32/libs/LibRTOSConsole/src/Console.c

4.3 ConsoleHandle Struct Reference

Collaboration diagram for ConsoleHandle:

Data Fields

- cmdState_t **cState**
- cspState_t **pState**
- TaskHandle_t **tHandle**
- int **cancel**
- struct {
  char **lines** [CONSOLE_LINE_HISTORY][CONSOLE_LINE_SIZE+CONSOLE_SAFETY_SPACE]
  int **lineHead**
  int **linePtr**
  } **history**

The documentation for this struct was generated from the following file:

- C:/HomeGit/STM32/libs/LibRTOSConsole/src/Console.c

4.4 cspState_t Struct Reference

Data Fields

- ctrlpStates_t **state**
- ctrlCodes_t **type**
- unsigned int **ptr**
- unsigned int **length**
- unsigned int **maxLength**
- char ∗ **buff**

The documentation for this struct was generated from the following file:

- C:/HomeGit/STM32/libs/LibRTOSConsole/src/Console.c

Chapter 5

File Documentation

5.1 C:/HomeGit/STM32/libs/LibRTOSConsole/conf/ConsoleConfig.h File Reference

This graph shows which files directly or indirectly include this file:

5.2 ConsoleConfig.h

Go to the documentation of this file.

```
/*
 * ConsoleConfig.h
 *
 *  Created on: Dec 9, 2024
 *      Author: Thorsten
 */



#ifndef INC_CONSOLE_CONSOLECONFIG_H_
#define INC_CONSOLE_CONSOLECONFIG_H_

#define CONSOLE_USERNAME  "STM32"

#define CONSOLE_USE_DYNAMIC_USERNAME 1

#define CONSOLE_LINE_HISTORY 8

#define CONSOLE_LINE_SIZE 120

#define CONSOLE_COMMAND_MAX_LENGTH 64

#define CONSOLE_HELP_MAX_LENGTH 512


#endif /* INC_CONSOLE_CONSOLECONFIG_H_ */
```

5.3 C:/HomeGit/STM32/libs/LibRTOSConsole/inc/Console.h File Reference

This graph shows which files directly or indirectly include this file:

Typedefs

- typedef struct ConsoleHandle ∗ ConsoleHandle_t
- typedef int(∗ CONSOLE_CommandFunc) (int argc, char ∗∗argv, void ∗context)

Functions

- ConsoleHandle_t CONSOLE_CreateInstance (unsigned int uxStackDepth, int xPrio)
- void CONSOLE_DestroyInstance (ConsoleHandle_t h)
- int CONSOLE_RegisterCommand (ConsoleHandle_t h, char ∗cmd, char ∗help, CONSOLE_CommandFunc func, void ∗context)

5.3.1 Typedef Documentation

5.3.1.1 CONSOLE_CommandFunc

```
typedef int(* CONSOLE_CommandFunc) (int argc, char **argv, void *context)
```

The CONSOLE_CommandFunc function pointer type is used to describe the arguments which are passed by the console processor toi the registered console function. The user gets the classical argc and argv arguments which are known well from a standard main in nearly any programming language. There is an additional context pointer, which optionally holds additional information. The pointer to this information is passed with the registration of the command by the user and can not be changed anymore at runtime

The return value of a console function is always zero on success or any non zero number when an error occured. The standard is more or less a non zero number which is smaller than 0.

5.3.1.2 ConsoleHandle_t

```
typedef struct ConsoleHandle* ConsoleHandle_t
```

The ConsoleHandle_t handle is an instance pointer of the console library which is generated whenever the CONSOLE_CreateInstance function returns with success.

5.3.2 Function Documentation

5.3.2.1 CONSOLE_CreateInstance()

```
ConsoleHandle_t CONSOLE_CreateInstance (
            unsigned int uxStackDepth,
            int xPrio)
```

The CONSOLE_CreateInstance function is used to create the console processor. There is no singleton pattern implemented for the console but as there is no stream abstraction for the console processor, it only makes sense to have one instance at runtime!

The return value of a console function is a null pointer in case an error occured or a pointer of type ConsoleHandle_t.

param uxStackDepth is the stack depth of the console processor thread in words param xPrio is the console processor priority. This should always be on a low level because the stdlib function may not block and so the processor always runs whenever it can. In case the reception of the newlib is blocking and interrupt based, the priority could be higher!

5.3.2.2 CONSOLE_DestroyInstance()

```
void CONSOLE_DestroyInstance (
             ConsoleHandle_t h)
```

The CONSOLE_DestroyInstance function is used to cleanup all used ressources which then stops the console processor. This leads to the case that no console functionallity is provided in the design anymore

5.3.2.3 CONSOLE_RegisterCommand()

```
int CONSOLE_RegisterCommand (
             ConsoleHandle_t h,
             char * cmd,
             char * help,
             CONSOLE_CommandFunc func,
             void * context)
```

The CONSOLE_RegisterCommand function is used to register custom commands which can be called by the console processor when the user enters the given command string. The arguments and an additional context pointer are passed by the console processor to the function as well.

param h is of type ConsoleHandle_t which is created by a call of CONSOLE_CreateInstance param cmd is of type char∗ which is the case sensitive name of the command param help is of type char∗ which is the help text or description of the command when the user types help param func is of type CONSOLE_CommandFunc which is the function pointer to the command param context is of type void∗ which is an optional data pointer which is passed to the function when called

5.4 Console.h

Go to the documentation of this file.

```
/*
 * Console.h
 *
 *  Created on: Dec 8, 2024
 *      Author: Thorsten
 */


#ifndef INC_CONSOLE_CONSOLE_H_
#define INC_CONSOLE_CONSOLE_H_

typedef struct ConsoleHandle* ConsoleHandle_t;


typedef int (*CONSOLE_CommandFunc)(int argc, char** argv, void* context);

ConsoleHandle_t CONSOLE_CreateInstance(  unsigned int uxStackDepth, int xPrio );

void CONSOLE_DestroyInstance( ConsoleHandle_t h );

int CONSOLE_RegisterCommand( ConsoleHandle_t h, char* cmd, char* help, CONSOLE_CommandFunc func, void*
context );



#endif /* INC_CONSOLE_CONSOLE_H_ */
```

5.5 C:/HomeGit/STM32/libs/LibRTOSConsole/src/Console.c File Reference

```
#include "FreeRTOS.h"
#include "task.h"
#include "semphr.h"
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <string.h>
#include <math.h>
#include <sys/queue.h>
#include "main.h"
#include "Console.h"
#include "ConsoleConfig.h"
```

Include dependency graph for Console.c:

Data Structures

- struct cmdEntry
- struct cmdState
- struct cspState_t
- struct ConsoleHandle

Macros

- #define **CONSOLE_SAFETY_SPACE** 4
- #define CHECK_FOR_OVERFLOW(x)
- #define xstr(a)
- #define str(a)

Typedefs

- typedef struct cmdEntry **cmdEntry_t**
- typedef struct cmdState **cmdState_t**

Enumerations

- enum **cspTYPE** { **csptNONE** , **csptCHARACTER** , **csptCONTROL** }
- enum **ctrlCodes_t** {
  ctrlC0_NUL = 0x00 , **ctrlC0_SOH** = 0x01 , **ctrlC0_STX** = 0x02 , **ctrlC0_ETX** = 0x03 ,
  ctrlC0_EOT = 0x04 , **ctrlC0_ENQ** = 0x05 , **ctrlC0_ACK** = 0x06 , **ctrlC0_BEL** = 0x07 ,
  ctrlC0_BS = 0x08 , **ctrlC0_TAB** = 0x09 , **ctrlC0_LF** = 0x0A , **ctrlC0_VT** = 0x0B ,
  ctrlC0_FF = 0x0C , **ctrlC0_CR** = 0x0D , **ctrlC0_SO** = 0x0E , **ctrlC0_SI** = 0x0F ,
  ctrlC0_DLE = 0x10 , **ctrlC0_DC1** = 0x11 , **ctrlC0_DC2** = 0x12 , **ctrlC0_DC3** = 0x13 ,
  ctrlC0_DC4 = 0x14 , **ctrlC0_NAK** = 0x15 , **ctrlC0_SYN** = 0x16 , **ctrlC0_ETB** = 0x17 ,
  ctrlC0_CAN = 0x18 , **ctrlC0_EM** = 0x19 , **ctrlC0_SUB** = 0x1A , **ctrlC0_ESC** = 0x1B ,
  ctrlC0_FS = 0x1C , **ctrlC0_GS** = 0x1D , **ctrlC0_RS** = 0x1E , **ctrlC0_US** = 0x1F ,
  ctrlC0_DEL = 0x7F , **ctrlC1_MASK** = 0x100 , **ctrlC1_DCS** = 0x101 , **ctrlC1_CSI** = 0x102 ,
  ctrlC1_SOS = 0x103 , **ctrlC1_PM** = 0x104 , **ctrlC1_APC** = 0x105 , **ctrlC1_ST** = 0x106 ,
  ctrlUNKNOWN = 0x1000 , **ctrlOVERFLOW** = 0x1001 }
- enum **ctrlpStates_t** {
  ctrlpsIDLE_DETECT = 0 , **ctrlpsSTART_C1** = 1 , **ctrlpsHANDLE_CSI** = 2 , **ctrlpsHANDLE_ST_1** = 3 ,
  ctrlpsHANDLE_ST_2 = 4 }

Functions

- cspTYPE **ControlSequenceParserConsume** (char input, cspState_t ∗s)
- static int **ProcessCommand** (char ∗command, int cmdLen, char ∗∗args, int numArgs, cmdState_t ∗c)
- static int **TransformAndProcessTheCommand** (char ∗lineBuff, int line_size, cmdState_t ∗cState)
- static void **PrintConsoleControl** (cspState_t ∗s)
- static int **ConsoleIsArrowLeft** (cspState_t ∗s)
- static int **ConsoleIsArrowRight** (cspState_t ∗s)
- static int **ConsoleIsArrowUp** (cspState_t ∗s)
- static int **ConsoleIsArrowDown** (cspState_t ∗s)
- static int **ConsoleIsEntf** (cspState_t ∗s)
- static void **PrintConsoleArrowLeft** (void)
- static void **ConsoleFunction** (void ∗arg)
- static int **ConsolePrintHelp** (int argc, char ∗∗argv, void ∗context)
- static int **ConsoleExecReset** (int argc, char ∗∗argv, void ∗context)
- static int **ConsolePrintKernelTicks** (int argc, char ∗∗argv, void ∗context)
- static int **ConsolePrintKernelVersion** (int argc, char ∗∗argv, void ∗context)
- static int **ConsoleWhoAmI** (int argc, char ∗∗argv, void ∗context)
- static int **ConsoleExit** (int argc, char ∗∗argv, void ∗context)
- static int **ConsoleMallInfo** (int argc, char ∗∗argv, void ∗context)
- static int **ConsoleGetEnv** (int argc, char ∗∗argv, void ∗context)
- static int **ConsoleSetEnv** (int argc, char ∗∗argv, void ∗context)
- static void **ConsoleRegisterBasicCommands** (ConsoleHandle_t h)
- ConsoleHandle_t CONSOLE_CreateInstance (unsigned int uxStackDepth, int xPrio)
- int CONSOLE_RegisterCommand (ConsoleHandle_t h, char ∗cmd, char ∗help, CONSOLE_CommandFunc func, void ∗context)
- void CONSOLE_DestroyInstance (ConsoleHandle_t h)

5.5.1 Macro Definition Documentation

5.5.1.1 CHECK_FOR_OVERFLOW

```
#define CHECK_FOR_OVERFLOW(
                x)
```

Value:

```
do { if ( ((x)+1) > s->maxLength ) \
{ s->type = ctrlOVERFLOW; s->length = 0; s->state = ctrlpsIDLE_DETECT; \
return csptCONTROL; } } while(0)
```

5.5.1.2 str

```
#define str(
                a)
```

Value:

```
#a
```

5.5.1.3 xstr

```
#define xstr(
                a)
```

Value:

```
str(a)
```

5.5.2 Function Documentation

5.5.2.1 CONSOLE_CreateInstance()

```
ConsoleHandle_t CONSOLE_CreateInstance (
            unsigned int uxStackDepth,
            int xPrio)
```

The CONSOLE_CreateInstance function is used to create the console processor. There is no singleton pattern implemented for the console but as there is no stream abstraction for the console processor, it only makes sense to have one instance at runtime!

The return value of a console function is a null pointer in case an error occured or a pointer of type ConsoleHandle_t.

param uxStackDepth is the stack depth of the console processor thread in words param xPrio is the console processor priority. This should always be on a low level because the stdlib function may not block and so the processor always runs whenever it can. In case the reception of the newlib is blocking and interrupt based, the priority could be higher!

5.5.2.2 CONSOLE_DestroyInstance()

```
void CONSOLE_DestroyInstance (
            ConsoleHandle_t h)
```

The CONSOLE_DestroyInstance function is used to cleanup all used ressources which then stops the console processor. This leads to the case that no console functionallity is provided in the design anymore

5.5.2.3 CONSOLE_RegisterCommand()

```
int CONSOLE_RegisterCommand (
            ConsoleHandle_t h,
            char * cmd,
            char * help,
            CONSOLE_CommandFunc func,
            void * context)
```

The CONSOLE_RegisterCommand function is used to register custom commands which can be called by the console processor when the user enters the given command string. The arguments and an additional context pointer are passed by the console processor to the function as well.

param h is of type ConsoleHandle_t which is created by a call of CONSOLE_CreateInstance param cmd is of type char∗ which is the case sensitive name of the command param help is of type char∗ which is the help text or description of the command when the user types help param func is of type CONSOLE_CommandFunc which is the function pointer to the command param context is of type void∗ which is an optional data pointer which is passed to the function when called

Index